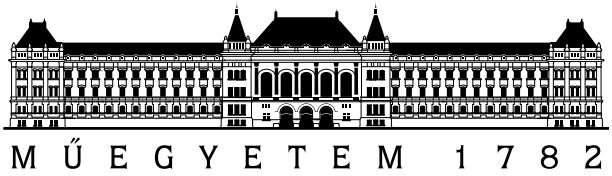


Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

Nyílt forráskódú Android alkalmazás közösségi fejlesztése

Témalaboratórium beszámoló

Főglein Simon István
ZA0D8T

Konzulens: Dr. Somogyi Péter

2022. december 4.

Tartalomjegyzék

Kivonat

A témalaboratóriumom során egy nyílt forráskódú Android alkalmazáshoz fejlesztettem töltőképernyő funkciót, hogy az az újabb, Android 12-t és frissebbet futtató rendszereken is megfelelően működjön. Ehhez részletesen megismerkedtem az Android dokumentációjával, példakódokat tanulmányoztam, többféle megoldást teszteltem különböző Android-verziókon. A munkám során betekintést nyertem a közösségi szoftverfejlesztésbe, ennek kihívásaiba, és egy nagy, több, mint egymillió felhasználó eszközén futó program hatalmas kódbázisába.

1. fejezet

Bevezetés, célkitűzések

Napjainkban egyre gyakrabban szembesülünk az egészséges életmód betartásának nehézségeivel. A Témalaboratórium tárgy keretein belül olyan szoftverfejlesztési feladattal szerettem volna foglalkozni, ami kapcsolódik a választott témához (orvosi informatika) és gyakorlati jelentősége is van, a szoftver használatával megkönnyíthetjük, jobbá tehetjük a felhasználók életét.

Ebben az irányban elindulva, számos különböző projektet megvizsgálva és a konzulensemmel egyeztetve döntöttem úgy, hogy a félév során az OpenFoodFacts[1] Android alkalmazását[2] fogom fejleszteni. Miután ez az elhatározás megszületett, igyekeztem olyan feladatot vállalni, ami megfelel a tárgy követelményeinek, tehát kellő mértékű kihívást jelent az implementációja, olyan részeket is tartalmaz, melyekkel új ismereteket szerezhetek, bővíthetem a meglévő tapasztalataimat, és mégis belefér a tárgy szűkös időkeretébe. Ezeket a szempontokat szem előtt tartva böngésztem a projekthez tartozó hibajelentéseket és egyéb felhasználói kéréseket, javaslatokat (ún. *feature request*-eket). Így bukkantam rá egy olyan hibajegyre (*GitHub Issue*), amelyben az alkalmazás töltőképernyőjének (splash screen) akkori implementációjának módosítását kérték, hogy az Android 31-es API szint feletti, azaz Android 12-t vagy annál újabbat futtató eszközökön is helyesen működjön. [4]

Mivel korábban az egyetemen már volt lehetőségem betekintést nyerni a mobilos szoftverfejlesztésbe, és régebbi Android verziót futtató készülékekhez már készítettem hasonló töltőképernyőt, továbbá sok alkalmazásnál használnak ilyen megoldást, érdekelt, hogy miben újították meg a fejlesztés folyamatát az új könyvtár bevezetésével.

A félév során tehát szerettem volna kipróbálni magam egy éles projektben, közelebbről megismerkedni az Androidos szoftverfejlesztéssel, továbbá bővíteni Kotlin nyelvű ismereteimet.

[1] Weboldal: `https://hu.openfoodfacts.org/`
[2] GitHub: `https://github.com/openfoodfacts/openfoodfacts-androidapp`

2. fejezet

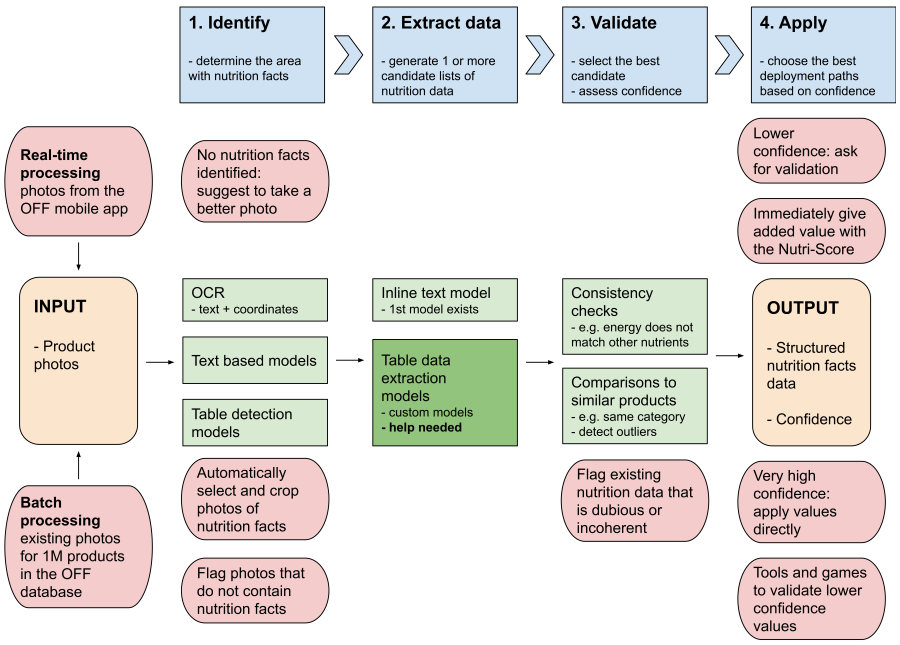
Az OpenFoodFacts projekt rövid áttekintése

Az OpenFoodFacts egy közösségi kezdeményezés, mely arra irányul, hogy minél több ember tájékozódhasson könnyen az élelmiszerek számos lényeges paraméteréről, így támogatva a tudatos étkezést. Az alkalmazás segítségével több, mint másfél millió élelmiszerről kaphatunk információt. Az ezekről elérhető fontosabb tulajdonságok a teljesség igénye nélkül:

- Összetevők
- Allergén információk
- Tápérték adatok
- Nutri-score[1]
- Környezetre gyakorolt hatás, ökolábnyom
- Feldolgozottsági szint

Az alkalmazásban elérhető információk többségét a felhasználók maguk tölthetik fel az oldalra. Ehhez használhatják például az OpenFoodFacts applikációt az okostelefonjukon, melynek segítségével beolvashatják egy adott élelmiszer vonalkódját, fotót készíthetnek a termékről, majd megadhatnak róla számtalan részletet. A program támogatja az optikai szövegfelismerést (Optical Character Recognition (OCR)), így akár a felhasználó időigényes közreműködése nélkül is ki tud nyerni lényeges információkat a feltöltött képekből (2.1 ábra). Az így felvett információk bekerülnek egy adatbázisba, lehetővé téve a program többi felhasználója számára az adatokhoz való szabad hozzáférést.

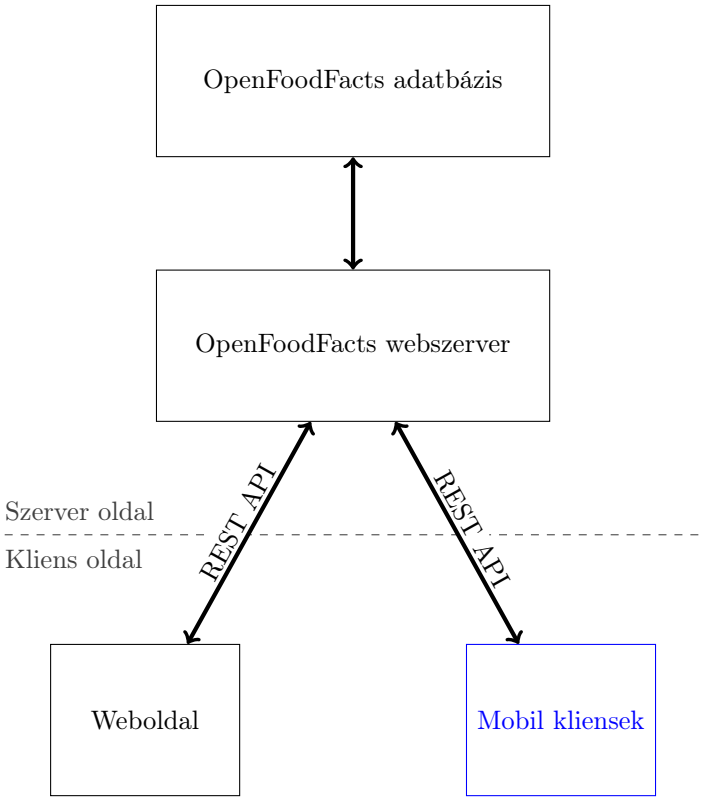
[1] Részletek a Nutri-score-ról: `https://hu.openfoodfacts.org/nutriscore`



2.1. ábra. Az OpenFoodFacts képes automatikusan is információkinyerésre a feltöltött képek alapján. Forrás: [7]

Az OpenFoodFacts natív Android és iOS applikáció mellett rendelkezik egy keresztplatformos, Flutter-alapú alkalmazással[2] is, továbbá egy webes alkalmazáson keresztül is hozzáférhetünk az élelmiszerek adataihoz. A kliensek REST API segítségével kommunikálhatnak a szerverrel. Az OpenFoodFacts architektúrájának sematikus rajza a 2.2 ábrán látható. A továbbiakban az architektúra szerver oldali komponenseivel nem foglalkozom bővebben, az ábrán kékkel jelölt mobilos kliensek közül is csak a natív Androidos alkalmazást mutatom be részletesebben.

[2] `https://github.com/openfoodfacts/smooth-app`



2.2. ábra. Az OpenFoodFacts rendszer architektúrája, a mobilos kliensek elhelyezkedése az architektúrában

3. fejezet

Androidos SplashScreen megoldások ismertetése

A továbbiakban ismertetett tevékenységek pontosabb megértéséhez elengedhetetlen az Androidon elérhető SplashScreen megoldások rövid áttekintése.

3.1. API level 30 (Android 11) és korábbi

Az Android 12-ben bevezetett új API előtt nem volt rendszerszintű támogatás töltőképernyők fejlesztésére, így a fejlesztők két lehetőség közül választhattak [2]:

- Egyedi téma bevezetése: a nézet *windowBackground* tagváltozóját változtatják meg, majd állítják vissza az alkalmazás betöltését követően a kívánt értékre
- Egyedi Activity létrehozása: ezzel egy dedikált osztályt és nézetet hoznak létre a töltőképernyő funkció megvalósítására, amely a betöltést vagy időtúllépést követően elindítja az alkalmazás főképernyőjét

Az OpenFoodFacts alkalmazásban az utóbbit választották a fejlesztők, ez viszont azt eredményezte, hogy Android 11 fölötti eszközökön két különböző töltőképernyő jelent meg az alkalmazás indulásakor, emiatt vált szükségessé az alkalmazás felkészítése az újabb verzióval való helyes működésre.

3.2. API level 31 (Android 12) és újabb

A SplashScreen API bevezetésével maga az operációs rendszer biztosít egységes megoldást töltőképernyő készítésére. Ez olyan lehetőségekkel bővítette a programozók eszköztárát, ami korábban nem, vagy csak körülményesen volt megvalósítható (pl. animált ikonok megjelenítése a töltőképernyőn). Az új API arra is lehetőséget ad, hogy a régebbi szoftververziót futtató eszközökön a SplashScreen compat könyvtár (mely az AppCompat könyvtár[1] részeként érhető el) segítségével biztosítsuk a visszafele kompatibilitást.

[1] `https://developer.android.com/jetpack/androidx/releases/appcompat`

4. fejezet

Jelentősebb állomások a fejlesztés során

Munkám jelentős részét az Android dokumentációjának tanulmányozása, illetve a különböző lehetőségek tesztelése tette ki. Fontos volt még megismernem a projekt fejlesztésére vonatkozó irányelveket (pl. kódformázási konvenciók, pull requestek pontos leírása) [6].

4.1. Ismerkedés a projekttel

A projekt meghatározását követően igyekeztem megismerni annak felépítését, kideríteni, hogy pontosan melyik modulokkal kell majd dolgoznom a töltőképernyő frissebb verziójának elkészítéséhez. Ehhez klónoztam a projekt GitHub repositoryját, és az Android Studio fejlesztőkörnyezet segítségével részletesen tanulmányoztam a projekt struktúráját, a releváns kódrészleteket, beszereztem a függőségeket, és lefordítottam a programot.

4.2. Migrálás az új API-ra, visszafelé kompatibilitás biztosítása

Android 12-es verziót futtató emulátor telepítését követően az Android fejlesztői dokumentációjában fellelhető migrációs útmutatóból tájékozódtam a további feladataimról [2]. Az itt leírtakat követve a töltőképernyő helyesen jelent meg az API 31-et futtató emulátoron. Ilyen funkció esetében azonban nem feledkezhetünk meg a visszafelé kompatibilitás támogatásáról sem, ezért szükséges volt az új könyvtárat nem ismerő eszközökön is tesztelni a programot. A régebbi API verziót futtató eszközök esetében azt a megoldást választottam, hogy a már meglévő töltőképernyő jelenjen meg (azaz egy külön splash screen Activity induljon el az alkalmazás inicializálásakor), míg az újakon a frissen implementált verzió. A funkció tesztelése során megállapítottam, hogy a megoldás alapvetően a megfelelő eredményt adja, azonban – ahogy az a GitHub issueban is szerepelt [4] – az éjszakai módban nem az elvárt módon működik: sötét helyett fehér háttérszínnel jelenik meg a képernyő. A hibajegyhez mellékelt képernyőképen szereplőhöz hasonló kinézetű töltőképernyőt az emulátoron nem

sikerült reprodukálni, ott az éjszakai mód bekapcsolását követően is fehér háttérszínnel indult el a töltőképernyő, csak az alkalmazás további részei használták az éjszakai témát. Mivel Android 12-t futtató eszköz nem állt rendelkezésemre, ezért valódi hardveren nem tudtam tesztelni az programot.

Kódrészlet 4.1. A SplashScreen compat könyvtár segítségével visszafele kompatibilitást támogató kódrészlet

```
val splashScreen: SplashScreen? =
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
        installSplashScreen()
    } else {
        null
    }
splashScreen?.setKeepOnScreenCondition { true }
```

A munkám során alkalmam nyílt megismerni egy, az Androidos szoftverfejlesztésben használt megoldást az alkalmazás különböző verzióinak egyszerű kezelésére és előállítására. Ebben a projektben ez különösképpen nagy jelentőséggel bír, hiszen nemcsak arról van szó, hogy az alkalmazást több alkalmazásboltban is terjesztik (pl. Google Play, F-Droid), hanem ugyanarra a kódbázisra három különböző alkalmazást (OpenFoodFacts (OFF), OpenBeautyFacts (OBF), OpenPetFoodFacts (OPFF)) is építenek. Ez azért is lehetséges, mert az alkalmazás lényegében csak egy felhasználói felület az OpenFoodFacts (és a többi verzió) adatbázisának elérésére. Így maga a kliens oldali alkalmazás az ikonok és egyéb felhasználói felületi elemek kivételével megegyezik a három verzió között. A build variants használata lehetővé teszi, hogy mindössze egyetlen Android appot készítsenek, amit különféle verziókra (flavor) lehet lefordítani. Eleinte nehézséget okozott, hogy alapértelmezetten a Git repository klónozását követően az OpenBeautyFacts alkalmazás települt. Mivel az alkalmazás szinte teljesen megegyezik az OpenFoodFacts alkalmazással, a tesztelés során ez nem jelentett gondot, de a splash screenen megjelenő ikon más, ezért az éjszakai mód teszteléséhez már az OpenFoodFacts-et volt szükséges telepíteni. Némi kutatómunka után találtam rá arra a lehetőségre Android Studioban, amivel a fordítandó build variant-ot lehet kiválasztani. Így a továbbiakban a Google Play-re targetelt OpenFoodFacts alkalmazással folytattam a fejlesztést.

4.3. Implementáció továbbfejlesztése: sötét mód támogatása

Az alapműködés implementációját követően a munkám az éjszakai módban látható töltőképernyő helyes megjelenítésére irányult. Sikerült kölcsönkérnem egy Android 12-es verziót futtató eszközt, azonban ezen sem sikerült előállítanom a GitHub issue-ban [4] szereplő módon a hibát, így folytattam az okának felkutatását a kódban és az interneten egyaránt. Rövid utánajárást követően kiderült, hogy a *styles.xml* fájlból nem volt megadva *night* erőforrásminősítővel (resource qualifier) ellátott verzió. Ezt a hiányosságot pótoltam, létrehoztam egy éjszakai módban helyesen megjelenő stílus erőforrásfájlt. Ebben a háttérszín megváltoztatásán túl szükséges volt a SplashScreen API-által biztosított témából egy

sajátot leszármaztatni, mely rendelkezik az ikon helyes megjelenítéséhez szükséges tulajdonságokkal. Az így született téma az alábbi:

```
<style name="SplashTheme"
  parent="Theme.SplashScreen.IconBackground">
    <item name="windowSplashScreenBackground">
        @color/grey_800
    </item>
    <item name="windowSplashScreenIconBackgroundColor">
        @color/grey_400
    </item>
    <item name="windowSplashScreenAnimatedIcon">
        @mipmap/ic_launcher_round
    </item>
</style>
```

Ez azonban nem jelentett megoldást, mert a téma, amiből az API 31 feletti helyes működésért le kell származtatnunk a saját témánkat, nem támogatja az AppCompatActivity használatát, ami az OpenFoodFacts activity-implementációjának ősosztálya. Így a SplashScreen API által biztosított téma kizárólagos használatával az alkalmazás a töltőképernyő megjelenítését követően hibára futott, ez a 4.1 ábrán látható.

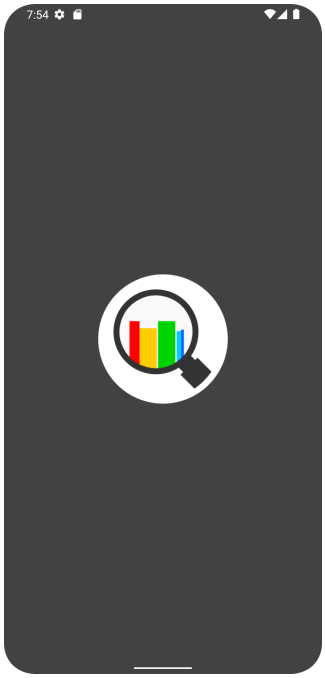
```
2022-10-30 19:43:00.107 8078-8078/? E/AndroidRuntime: FATAL EXCEPTION: main
    Process: openfoodfacts.github.scrachx.openfood.debug, PID: 8078
    java.lang.RuntimeException: Unable to start activity ComponentInfo{openfoodfacts.github.scrachx.openfood
     .debug/openfoodfacts.github.scrachx.openfood.features.splash.SplashActivity}: java.lang
     .IllegalStateException: You need to use a Theme.AppCompat theme (or descendant) with this activity.
```

4.1. ábra. AppCompat téma használata nélkül az alkalmazás összeomlik

A probléma megoldására az Android fejlesztői oldalán elérhető leírás nyújtott iránymutatást [2]. A *postSplashScreenTheme* tagváltozó megfelelő beállítását követően az alkalmazás már megfelelően működött, a töltőképernyő után a főképernyő helyesen jelent meg. Így az éjszakai módban használt téma kódja az alábbi:

```
<style name="SplashTheme"
  parent="Theme.SplashScreen.IconBackground">
    <item name="windowSplashScreenBackground">
        @color/grey_800
    </item>
    <item name="windowSplashScreenIconBackgroundColor">
        @color/grey_400
    </item>
    <item name="postSplashScreenTheme">
        @style/Theme.AppCompat.DayNight
    </item>
    <item name="windowSplashScreenAnimatedIcon">
        @mipmap/ic_launcher_round
    </item>
</style>
```

A fent definiált téma az alábbi módon jelenik meg a gyakorlatban (4.2 ábra):



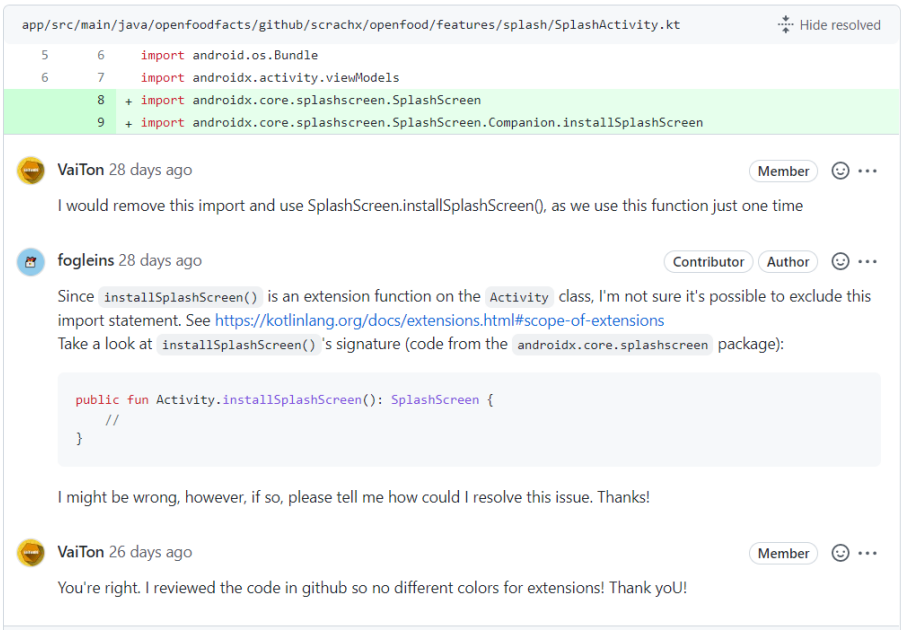
4.2. ábra. A töltőképernyő éjszakai módban, Android 12-n

4.4. Egyezetés a projekt fejlesztőivel

A funkciók tesztelését követően kommunikációt folytattam az alkalmazás fejlesztőivel a pull request lehetséges javításairól, illetve a módosításaim kódbázisba kerüléséről [8]. Az alkalmazás egyik fejlesztője egy import eltávolítását javasolta, rövid utánajárást követően [5] én viszont arra jutottam, hogy ez nem hagyható el, ugyanis egy ún. Kotlin *extension function*-ről van szó, amire az import nélkül nem tudunk hivatkozni. A Kotlinban támogatott extension function-ök olyankor lehetnek hasznosak, amikor egy már – nem általunk – megírt osztály lehetőségeit szeretnénk bővíteni. Ilyen megoldást alkalmaztak a SplashScreen könyvtár [3] fejlesztői is, az Android által biztosított Activity osztályt bővítették ki egy *installSplashScreen* metódussal:

Kódrészlet 4.2. Részlet a SplashScreen könyvtár kódjából az extension function-nel

```
public fun Activity.installSplashScreen():
        SplashScreen {
    // ...
}
```



4.3. ábra. Egyeztetés a projekt egyik fejlesztőjével az extension function importjáról

Végül a projekt másik fejlesztője is az enyémmel azonos következtetésre jutott, az import nem hagyható el, így a módosításaimat elfogadták, és bekerültek az alkalmazás kódbázisába.

5. fejezet

Összegzés

A Témalaboratórium tárgy keretein belül alkalmam nyílt betekintést nyerni egy élő, aktívan fejlesztés alatt álló, széles felhasználói körrel rendelkező projekt fejlesztésébe. Ezalatt számos új ismeretre tettem szert: ízelítőt kaptam, hogy hogyan lehet kiigazodni egy összetett szoftver forrásfájljai között, hogyan lehetséges olyan emberekkel közösen dolgozni egy ilyen kihíváson, akikkel sosem találkoztam, és több ezer kilométerre élnek tőlem. Megtanultam továbbá, hogy hogyan lehetséges egy alkalmazás forráskódját más hasonló szoftverben is újrafelhasználni, mi a különböző alkalmazásboltok és ezek sajátosságainak kezelésének módja. Ezen kívül korábbi tapasztalataimat is bővítettem: jobban elmélyültem a Kotlin nyelv által nyújtott eszközök, modern nyelvi elemek használatában (pl. extension function-ök), lehetőségem volt szélesebb körben megismerkedni az Androidos szoftverfejlesztéssel, valamint áthatóan megismertem néhány könyvtárat (pl. SplashScreen, AppCompat).

A féléves munkám pedig a megszerzett tudáson túl is kifizetődött, ugyanis az általam készített töltőképernyő-implementáció bekerült az alkalmazás kódbázisába, így több százezer felhasználó találkozhat vele nap mint nap a program indítása során.

Szójegyzék

Activity Android alkalmazáskomponens, mely rendelkezik felhasználói felülettel, felhasználói interakciót tesz lehetővé. A hivatalos dokumentáció szerint: ”An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with“ [1]. 5, 6, 8, 9

Android Studio Az Android szoftverfejlesztés hivatalos fejlesztőkörnyezete. 6, 7

AppCompat Lehetőséget ad új API-verziók használatára régebbi Android-verziókon. 5, 8, 11

emulátor Számítógépes program vagy hardver, ami más programoknak vagy eszközöknek a környezetét (vagy annak részét) szimulálja, vagyis lehetővé teszik az adott rendszerrel nem kompatibilis programok (vagy operációs rendszerek) vagy számítógépek futtatását. Android fejlesztésben szinte mindig egy virtuális telefont jelent [9]. 6

erőforrásminősítő Az Android platform által támogatott megoldás az eszközkonfigurációnak legmegfelelőbb erőforrások (pl. képek, képernyőelrendezések) betöltésére. 7

Git Széleskörben elterjedt verziókezelő rendszer elsősorban szoftverfejlesztéshez, forráskódfájlok verzióinak kezelésére. 7

GitHub Verziókövetési- és közösségi szoftverfejlesztési lehetőségeket (pl. hibajegyek kezelése) biztosító szolgáltatás. `https://github.com`. 1, 6, 7

repository Itt: Git repository, azaz verziókövetéssel ellátott forráskódtároló. Általában egy projekt teljes forrásállománya található benne, így a letöltését és a függőségek beszerzését követően lefordíthatjuk a szoftvert. 6, 7

SplashScreen Töltőképernyő, mely az alkalmazás indításakor látható amíg a főképernyő nem tölt be. 1, 5–9, 11

téma Az alkalmazás megjelenését befolyásoló, előre definiált megjelenési beállítások. 5, 8, 9

Betűszavak

API Application Programming Interface. 1, 5–8, 12

OBF OpenBeautyFacts. 7

OCR Optical Character Recognition. 2

OFF OpenFoodFacts. 1–5, 7, 8

OPFF OpenPetFoodFacts. 7

Irodalomjegyzék

[1] Android API documentation: *Activity class reference*. `https://developer.android.com/reference/android/app/Activity`. Version: 2022. – [Online; letöltve: 2022. november 30.]

[2] Android API documentation: *Migrate your existing splash screen implementation to Android 12 and higher*. `https://developer.android.com/develop/ui/views/launch/splash-screen/migrate`. Version: 2022. – [Online; letöltve: 2022. november 29.]

[3] Android API documentation: *SplashScreen class reference*. `https://developer.android.com/reference/kotlin/androidx/core/splashscreen/SplashScreen`. Version: 2022. – [Online; letöltve: 2022. november 29.]

[4] Hibajegy: *Implement Android 12 new splashscreen API (4548)*. `https://github.com/openfoodfacts/openfoodfacts-androidapp/issues/4548`. Version: 2022. – [Online; letöltve: 2022. november 29.]

[5] Kotlin language documentation: *Scope of extension functions*. `https://kotlinlang.org/docs/extensions.html#scope-of-extensions`. Version: 2022. – [Online; letöltve: 2022. november 29.]

[6] OpenFoodFacts developer documentation: *Join the development*. `https://github.com/openfoodfacts/openfoodfacts-androidapp/blob/develop/CONTRIBUTING.md`. Version: 2022. – [Online; letöltve: 2022. december 4.]

[7] OpenFoodFacts Wiki: *Nutrition facts table data extraction*. `https://wiki.openfoodfacts.org/Nutrition_facts_table_data_extraction`. Version: 2022. – [Online; letöltve: 2022. november 30.]

[8] Pull request: *Use new SplashScreen API on Android 12 and newer (API level 31 and up) (4871)*. `https://github.com/openfoodfacts/openfoodfacts-androidapp/pull/4871`. Version: 2022. – [Online; letöltve: 2022. november 29.]

[9] Wikipédia: *Emulátor*. `https://hu.wikipedia.org/wiki/Emul%C3%A1tor`. Version: 2022. – [Online; letöltve: 2022. december 4.]